Ananya Gaikwad- Data Analyst

Pariksha Mitra - Data Architecture Project Overview

Project Overview

Pariksha Mitra is an educational platform designed for Marathi medium students from 5th to 10th grade. The project aims to provide personalized, data-driven educational experiences by leveraging modern data architecture for effective student profiling, performance tracking, and curriculum support. The core of the project is its data architecture, which ensures the system is scalable, efficient, and capable of providing real-time educational insights.

As a **Data Analyst**, your role focuses on the data architecture of the system, particularly around storage, retrieval, and utilization. This involves implementing data models, processing algorithms, and storage strategies that align with the educational goals of the platform while ensuring scalability, data privacy, and system performance.

1. Data Modelling

Key Models

The data modelling for Pariksha Mitra focuses on creating several key entities and their relationships. The following models will be defined and structured to meet the platform's needs:

1. **Student Profiles**: Each student will have a profile that includes personal details, academic history, performance data, and preferences.
2. **Chapter Structure**: The curriculum will be broken down into chapters and subtopics. This structure will enable tracking of progress, test assignments, and chapter-based analytics.
3. **Exercise Definitions**: Each exercise or assignment will be stored with its specific requirements, questions, and answer types.
4. **Question Bank**: A dynamic and structured collection of questions related to the chapters. It will include metadata about difficulty, type, and learning objectives.
5. **Test Results**: Every student’s results from tests and quizzes will be captured, enabling performance tracking over time.
6. **Performance Analytics**: A layer of analytics that tracks a student’s performance, strengths, weaknesses, and provides actionable insights for

personalized learning.

Code Implementation (Python with SQLAlchemy)

For creating these models, Python with SQLAlchemy is used for defining the schema and setting up the relationships between the entities. Here's an example for defining the **Student** model:

python

```
from sqlalchemy import Column, Integer, String, DateTime, Float
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base
Base = declarative_base()
class Student(Base):
   __tablename__ = 'students'

   id = Column(Integer, primary_key=True)
   first_name = Column(String(50), nullable=False)
   last_name = Column(String(50), nullable=False)
   grade = Column(Integer, nullable=False)
   medium = Column(String(20), default='Marathi')
   date_of_birth = Column(DateTime)
test_results = relationship('TestResult', back_populates='student')
```

Relationships between Entities

The relationships between different entities are crucial for ensuring data integrity and ease of retrieval. The **Data Relationship Diagram** can help visualize these connections. Here's an example:

mermaid

```
erDiagram
   STUDENT ||--o{ TEST_RESULT : "takes"
   STUDENT ||--o{ PERFORMANCE_ANALYTIC : "has"
   CHAPTER ||--o{ EXERCISE : "contains"
   CHAPTER ||--o{ QUESTION_BANK : "includes"
   EXERCISE ||--o{ TEST_RESULT : "generates"
   EXERCISE ||--|| QUESTION_BANK : "uses"
```

```
EXERCISE ||--|| QUESTION_BANK : "uses"
```

This diagram shows that students take tests, generate results through exercises, and receive performance analytics based on their overall progress.

2. Data Storage Strategy

A robust storage strategy is necessary for efficiently managing large amounts of educational data. The following considerations will guide the storage decisions:

1. Database Structures with Efficient Indexing

- For structured data such as student profiles and test results, **PostgreSQL** will be used. This relational database supports complex queries and efficient indexing for fast retrieval.
- For unstructured or semi-structured data, such as exercises, question banks, and performance analytics, **MongoDB** (NoSQL) will be employed for its flexibility in handling large amounts of data with varying formats.

2. Data Relationship Diagrams

The data relationship diagram (shown above) will be used to visualize the relationships between different entities, ensuring efficient data retrieval and helping define the primary and foreign keys that ensure relational integrity.

3. Data Privacy and Security Measures

To ensure the security and privacy of sensitive student data, several measures will be implemented:

- **Encryption**: Data will be encrypted both at rest and in transit using industry-standard encryption protocols (e.g., AES-256, SSL/TLS).
- **Access Control**: Role-based access control (RBAC) will be implemented to ensure that only authorized users (e.g., administrators, teachers) can access certain datasets.
- **Data Anonymization**: In some cases, sensitive personal data will be anonymized for analytics purposes.

3. Data Processing

The data processing component focuses on transforming raw data into meaningful insights. This includes the development of algorithms to enhance the platform's functionality and provide personalized learning experiences.

1. Question Randomization

The question randomization algorithm ensures that each student is presented with a unique set of questions each time they attempt an exercise or test. This

helps prevent cheating and provides a personalized learning experience.
Example pseudocode for randomization:

python

```
import random
def randomize_questions(question_bank, num_questions=10):
    random.shuffle(question_bank)
    return question_bank[:num_questions]
```

2. Performance Calculation

Performance is calculated by evaluating test results against a predefined grading scale. Each test result is compared to the student's previous performance, and trends are established over time.
Sample pseudocode for performance calculation:

python

```
def calculate_performance(test_results, max_score):
    total_score = sum(test_results)
    performance_percentage = (total_score / max_score) * 100
    return performance_percentage
```

3. Analytics Generation

Analytics will be generated for students, teachers, and administrators to track progress. The analytics system will use data processing to identify strengths, weaknesses, and suggest personalized learning paths for students.
Sample pseudocode for analytics generation:

python

```
def generate_student_analytics(student_data):
    performance_metrics = calculate_performance(student_data['test_results'],
max_score=100)
    strengths, weaknesses =
identify_strengths_weaknesses(student_data['test_results'])
    return {
        'performance_percentage': performance_metrics,
```

```
    return {
        'performance_percentage': performance_metrics,
        'strengths': strengths,
        'weaknesses': weaknesses
    }
```

4. NoSQL Schema Designs

To accommodate the flexible nature of student data, MongoDB will be used to store documents such as user profiles, test logs, and performance analytics. A typical schema for student profiles might look like:

json

```
{
  "_id": ObjectId,
  "student_id": "12345",
  "profile": {
    "name": "John Doe",
    "grade": 7,
    "school": "XYZ School",
    "medium": "Marathi"
  },
  "test_results": [
    {
      "test_id": "67890",
      "score": 80,
      "date": "2024-01-15"
    }
  ]
}
```

5. Performance Optimization Recommendations

To ensure that the system can handle a large number of concurrent users and provide real-time performance analytics, several optimization strategies will be employed:

- **Database Sharding**: Distribute data across multiple servers to balance the load and improve query performance.
- **Redis Caching**: Frequently accessed data, such as test results and student profiles, will be cached in **Redis** to reduce database load and improve response times.
- **Connection Pooling**: Manage database connections efficiently to prevent performance degradation due to frequent connection opening/closing.
- **Read Replicas**: Implement read replicas for horizontal scaling to offload read-heavy queries from the primary database.
- **Prepared Statements**: Use prepared statements to minimize SQL injection risks and optimize query execution.

Conclusion

This data architecture plan provides a comprehensive roadmap for implementing the backend infrastructure of the **Pariksha Mitra** platform. By combining relational and NoSQL databases, employing efficient data processing algorithms, and ensuring high system performance, the project will be able to handle large-scale data while delivering real-time educational insights to students.

Through clear and concise data modelling, optimized storage solutions, and privacy-focused design, **Pariksha Mitra** will not only cater to the needs of the students but also scale seamlessly as the user base grows.